AFCRL - 72-0109

AD 740849

UNIVERSAL FUNCTION THEORY AND GALOIS LOGIC STUDIES

James T. Ellison

Defense Systems Division
Univac Division of Sperry Rand Corporation
Univac Park P. O. Box 3525, St. Paul, Minnesota 55101

Contract No. F19628 - 71 - C - 0208 NEW
Project No. 5632
Task No. 563207
Work Unit No. 56320701

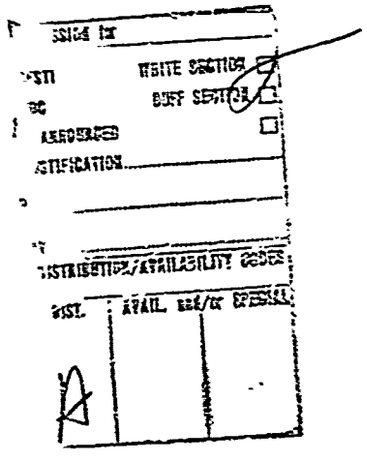
FINAL REPORT
Report Period: 1 June 1971 thru 31 January 1972
March 1972

Contract Monitor: Hans H. Zschirnt
Data Sciences Laboratory

Prepared
for

AIR FORCE CAMBRIDGE RESEARCH LABORATORIES
AIR FORCE SYSTEMS COMMAND
UNITED STATES AIR FORCE
BEDFORD, MASSACHUSETTS 01730

Qualified requestors may obtain additional copies from the Defense Documentation Center. All others should apply to the Clearinghouse for Federal Scientific and Technical Information.



AFCRL - 72-0109

UNIVERSAL FUNCTION THEORY AND GALOIS LOGIC STUDIES

James T. Ellison

Defense Systems Division
Univac Division of Sperry Rand Corporation
Univac Park P. O. Box 3525, St. Paul, Minnesota 55101

Contract No. F19628 - 71 - C - 0208
Project No. 5632
Task No. 563207
Work Unit No. 56320701

FINAL REPORT
Report Period: 1 June 1971 thru 31 January 1972
March 1972

Contract Monitor: Hans H. Zschirnt
Data Sciences Laboratory

Prepared
for

AIR FORCE CAMBRIDGE RESEARCH LABORATORIES
AIR FORCE SYSTEMS COMMAND
UNITED STATES AIR FORCE
BEDFORD, MASSACHUSETTS 01730

ABSTRACT

This report advances work in two areas relevant to logic design in an MSI/LSI technology. First, the theory of universal functions is advanced to the full generality of finite mathematical structures. A universal function is a function which, by appropriate parametrization of a fixed subset of variables, becomes an arbitrary preselected function of its remaining variables. A sequence of theorems are proven which establishes the character, interrelationship and means of synthesis of universal functions for all finite mathematical structures, including all possible settings for logic design, culminating in the characterization of universal functions as the universal elements of an appropriately defined functor between two categories. Second, the theory of Galois logic design is enhanced in various ways. Various Galois lattice-like operations are defined and compared. Next, methods of converting Galois multiplication gates to binary addition gates and to Galois linear gates are described. Finally, techniques are offered utilizing the Galois linear module to reduce hardware at the cost of switching speed.

Unclassified
Security Classification

DOCUMENT CONTROL DATA - R&D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)*
Univac Division of Sperry Rand Corporation
Defense Systems Division
Univac Park, P. O. Box 3525. St. Paul, Minnesota 55165

2a. REPORT SECURITY CLASSIFICATION
Unclassified

2b. GROUP

3. REPORT TITLE

UNIVERSAL FUNCTION THEORY AND GALOIS LOGIC STUDIES

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*
Scientific. Final. 1 June 1971 - 31 January 1972 Approved 18 February 1972

5. AUTHOR(S) *(First name, middle initial, last name)*

James T. Ellison

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
March 1972	57	10

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
F19628-71-C-0208	
b. PROJECT, TASK, WORK UNIT NOS. 5632-07-01	
c. DOD ELEMENT 61102F	9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*
d. DOD SUBELEMENT 681305	AFCRL-72-0109

10. DISTRIBUTION STATEMENT

A - Approved for public release; distribution unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
TECH, OTHER	Air Force Cambridge Research Laboratories (LR) L. G. Hanscom Field Bedford, Massachusetts 01730

13. ABSTRACT

This report advances work in two areas relevant to logic design in an MSI/LSI technology. First, the theory of universal functions is advanced to the full generality of finite mathematical structures. A universal function is a function which, by appropriate parametrization of a fixed subset of variables, becomes an arbitrary preselected function of its remaining variables. A sequence of theorems is proven which establishes the character, interrelationship and means of synthesis of universal functions for all finite mathematical structures, including all possible settings for logic design, culminating in the characterization of universal functions as the universal elements of an appropriately defined functor between two categories. Second, the theory of Galois logic design is enhanced in various ways. Various Galois lattice-like operations are defined and compared. Next, methods of converting Galois multiplication gates to binary addition gates and to Galois linear gates are described. Finally, techniques are offered utilizing the Galois linear module to reduce hardware at the cost of switching speed.

DD FORM 1 NOV 65 1473

Unclassified
Security Classification

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
MSI/LSI						
Logic Design						
Galois Fields						
Galois Logic						
Boolean Logic						
Iterative Logic Nets						
Computer Organization						
Universal Functions						

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

SECTION I

INTRODUCTION

Both universal function theory and the Galois approach to logic design are attempts to escape a paradox created by the rapid advances in electronic packaging of recent years. The advent of medium-to-large scale integration has reduced the cost of individual electronic components (e.g., diodes, transistors and resistors) to the vanishing point while at the same time greatly increasing the first-time cost of producing a new logic block (e.g., MSI/LSI chip types). The traditional approach to special circuit design was oriented to minimizing diode/transistor-counts, but this is counterproductive in a technology in which the value such savings is negligible and forces one to design an entirely new, uniterative, special-purpose logic block.

The idea of creating logic blocks capable of generating arbitrary functions of a fixed set of variables attacks the problem of multiplying new logic module types by enabling a single logic block to serve in a great variety of capacities. The universal modules of [1] and [2], for example, and recent read only memories (ROMs) offer the extreme of this sort of capability. The theory of such modules, initially developed in [3] and extended in [4], now appears in full generality in Section Two.

The paradox above suggests that traditional optimization methodology minimizes the wrong things in view of new technology. To be effective, network simplification and reduction must take place subject to the constraint that the existing set of basic logic blocks--the MSI/LSI chip types--need not be increased. This is the major goal and advantage of Galois logic design. Instead of the traditional Boolean operations like AND, OR, NOT, NAND, etc., representing a few transistors and diodes, the

Galois approach utilizes operations like Galois multiplication and the Galois linear function which represent whole MSI/LSI chips. Algebraic reduction in Galois logic design thus reduces chip count without affecting chip design with the same effect as algebraic reduction in Boolean logic design, which reduces diode-transistor count without affecting the internal design of the individual Boolean gate.

A year ago research described in [ ] offered a methodology for internal optimization of individual Galois gate and also discussed a variety of approaches to generating universal Galois modules. The Galois primitives most seriously considered there are Galois addition, Galois multiplication and the Galois linear function. Section Three of this report investigates several operations on a Galois field which act in a manner similar to the Boolean AND-OR-NOT primitives.

Section Four discusses two special methods of Galois network design. The first is a special technique for enhancing a standard Galois multiplication gate to perform ordinary binary addition. The second systematically converts a standard Galois multiplication gate to a Galois linear module in an optimal way.

Section Five utilizes trade-offs between switching speed and hardware to design and uses generalized linear modules. The sixth and final section recommends detailed investigation and development of global network reduction algorithms as the next and final major step in making Galois theory a practical approach to logic design.

1.1. Basic Concepts of Finite Field Theory.

The simplest field of all is the two-element Boolean algebra {0, 1} under the Boolean operations of EXCLUSIVE OR and AND. The following is a formal definition of a field and the behavior of the field operations of addition, multiplication and inversion.

A field is a set S of elements and a pair of two-place operations + and · with the following properties:

i. closure: for all x and y in S, (x + y) and (xy) are in S;

ii. associativity: for all x, y and z in S,
$(x + y) + z = x + (y + z)$, and $(xy)z = x(yz)$;

iii. commutativity: for all x and y in S,

$$x + y = y + x, \text{ and } xy = yx;$$

iv. distributivity: for all x, y and z in S,

$$x(y + z) = xy + xz;$$

v. identities: there exist an additive identity, zero, and a multiplicative identity, one, in S such that, for all x in S,

$$x + 0 = x, \text{ and } x \cdot 1 = x;$$

vi. inverses: for each x in S, there exists a unique element y in S such that

$$x + y = 0,$$

and, for each nonzero x in S, there exists a unique element y in S such that

$$xy = 1.$$

A Galois field is a field with a finite number of elements.

The order of a finite field is the number of members it has. If p is a prime and n is any positive integer, there is a Galois field of order p^n. Any two Galois fields of the same order are isomorphic, so for practical purposes there is just one Galois field with p^n elements. This field is designated by $GF(p^n)$. Of primary concern will be with fields having 2^n elements. It is therefore convenient to let G_n represent $GF(2^n)$ and call n the power of the field. Define n^* as $2^n - 1$.

Because Galois fields are fields, all the usual operations of elementary algebra carried out in the usual ways give correct results. Moreover their finiteness permits further, very useful simplification. In fields G_n every element is its own additive inverse; that is,

$$x + x = 0. \tag{1-1}$$

This can be viewed as a generalization of the EXCLUSIVE OR operation in the two-element Boolean algebra G_1. As a result, in such fields the expression $(x + y)^2$ can be rewritten $x^2 + xy + xy + y^2$, whence

$$(x + y)^2 = x^2 + y^2. \tag{1-2}$$

For each x in G_n, it is clear that the sequence $(x, x^2, x^3, \ldots)$ must eventually begin to repeat, since G_n has only 2^n distinct elements. The order of an element x in G_n is the number of terms in the sequence $(x, x^2, \ldots)$ before it begins to repeat. Every field G_n has elements of order n^*. These elements are called generators of the field. Zero is the only element omitted from the sequence of powers of a generator. Let G_n' be the set of nonzero members of G_n. Every element of G_n' can be given a logarithmic representation as a unique power less than 2^n of a generator. The order of an element of G_n is always a divisor of n^*, the number of members of G_n'. From this it can be determined that the 2^nth member of the sequence generated by any element x of G_n must be x itself. Thus,

$$x^{2^n} = x,$$

from which it follows that, if x is nonzero,

$$x^{n^*} = 1, \text{ and}$$
$$x^{-1} = x^{2^n-2}.$$

There is a second method of characterizing elements of G_n. There exist n elements $\omega_0, \ldots, \omega_{n-1}$, called a basis for G_n, such that every element of G_n can be expressed in the form

$$x_0\omega_0 + \ldots + x_{n-1}\omega_{n-1},$$

where each x_i, $i = 0, \ldots, n-1$, is zero or one. From this point of view, G_n is an n-dimensional vector space. Thus, relative to a fixed basis $(\omega_0, \ldots, \omega_{n-1})$, each element x of G_n has a unique coordinate representation $(x_0, \ldots, x_{n-1})$. By the ordinary operations of algebra,

$$(x_0\omega_0 + \ldots + x_{n-1}\omega_{n-1}) + (y_0\omega_0 + \ldots + y_{n-1}\omega_{n-1}) =$$

$$(x_0 \oplus y_0)\omega_0 + \ldots + (x_{n-1} \oplus y_{n-1})\omega_{n-1}.$$

Thus, coordinate representation of the sum is the termwise mod-two sum of the coordinate representations.

It would be ideal for Galois logic design if there were a simple and direct relationship between logarithmic representation of a Galois field and coordinate representation. Unfortunately, this ideal can be approached with only varying degrees of success. The substance of much of [5], dealing with Galois addition and multiplication gate synthesis, is a search for reasonably compatible logarithmic and coordinate representations of the same Galois field.

The familiar fields of analysis have an enormously rich variety of functions which can be defined on them. Every function on a Galois field, however, turns out to be a polynomial of degree less than the number of elements in the field. For every one-variable function f defined in G_n, there exist elements $\alpha_0, \ldots, \alpha_n{}^*$ in G_n such that

$$f(x) = \alpha_0 + \alpha_1 x + \ldots + \alpha_n{}^* x^{n^*}.$$

This fact places an upper bound on the degree of complexity that need be considered for any logical function to be synthesized. Furthermore, in actual Galois synthesis, logic designers need operate only with simple polynomials which, at worst, behave in a manner identical to those of elementary algebra.

1.2. Miscellaneous Terminology.

Some familiarity with two-valued Boolean algebra is assumed. The two values are the constants zero and one. The same numerals are used to represent the additive and multiplicative identities of Galois fields, but their context will eliminate any essential ambiguity. Other Galois constants do not have a standardized representation but will be given representations relative to particular bases or generators.

The primitive operations or primitives of a Boolean algebra or a Galois field are those basic operations in terms of which all other functions on the structure can be defined. (In the literature, generators

of a Galois field are usually called primitives, but this usage is not followed here.) A formula is a (well-formed) expression consisting of constants or variables joined by primitive operations. A formula is a representation of the function indicated by its table of values. A k-variable function is a function with k arguments and can be represented by a formula in k variables. The function obtained by replacing a subset of its variables with constants is a parametrization of the function. A Boolean translator is a sequence of Boolean functions of the same variables.

Set membership, union, and intersection are indicated in the usual way: $x \in S$, $S \cup T$, $S \cap T$. A sequence of individuals $(s_0, \ldots, s_{k-1})$ is symbolized "$\underline{s}$", with $|\underline{s}|$ defined as the length k of the sequence. Similarly, a sequence of functions $(f_0(\underline{x}), \ldots, f_{k-1}(\underline{x}))$ is written $\underline{f}(\underline{x})$. Let $\underline{f}$ be j-argument function sequence of length k, and let $\underline{g}$ be k-argument function sequence of length m. Then $\underline{g} \cdot \underline{f}$ is the j-argument function sequence $\underline{h}$ of length m such that, for all $\underline{x}$,

$$h_i(\underline{x}) = g_i(\underline{f}(\underline{x})), \quad i = 0, \ldots, m-1.$$

A positive integer m underlined represents the set:

$$\underline{m} = \{0, 1, \ldots, m-1\}.$$

A set S written with a positive integral exponent k represents the set of all k-length sequences of members of that set. For example, $\underline{2}^5$ is the set of all five-bit sequences. If S and T are sets, the Cartesian product S x T is the set of all ordered pairs (s, t) with s in S and t in T. The weight w(b) of a vector b of binary values is its number of nonzero components. A unit vector is a binary sequence of weight one. The ones vector of length n is the sequence of n ones. If V is a set of vectors, w(V), the weight of V, is the sum of the weights of the members of V. If k is a natural number, then $\underline{B}(k)$ is the reverse binary representation of k; i.e., $\underline{B}$ is the function sequence such that

$$k = \sum_{0}^{n-1} 2^i B_i(k).$$

A natural number is a nonnegative integer. A number is dyadic if it is a power of two. A dyadic power of x is therefore of the form x^{2^i}, for some integer i. The symbols $\lceil r \rceil$ and $\lfloor r \rfloor$ represent the smallest integer not less than r and the integral part of r, respectively, for r a real number. If r is an integer,

$$\lceil r \rceil = \lfloor r \rfloor = r;$$

otherwise,

$$\lceil r \rceil = 1 + \lfloor r \rfloor.$$

A function f defined on a set S with values in a set T is indicated by

$$f: S \longrightarrow T.$$

If each member of T is the image under f of some member of S, then f is surjective. If each member of T is the image under f of at most one member of S, then f is injective. The function $f: S \longrightarrow T$ is bijective if it is both injective and surjective; this is indicated by writing

$$f: S \longleftrightarrow T.$$

Boolean variables will be written in small Roman type, and they will usually appear subscripted.

Galois variables may be either Roman or Greek lower case literals but will be subscripted only as Greek letters unless otherwise specified. These conventions permit a coordinate representation of a Galois variable x to be indicated by "$\underline{x}$", a sequence of Boolean variables. Analogous conventions will exist for Boolean and Galois function symbols and function sequences.

Let $\underline{x}$ be a sequence of length k and $\underline{y}$ be a sequence of length 2^k. The Boolean function $u(\underline{x}, \underline{y})$ is universal for $\underline{x}$ if each of the 2^{2^k} functions of $\underline{x}$ occur just once among the various parametrizations of u obtained by replacing $\underline{y}$ by a sequence of Boolean constants. The active

variables of $u(\underline{x}, \underline{y})$ are $\underline{x}$, while its selection variables are $\underline{y}$. A universal Galois function is a Galois function $u(\underline{\xi}, \underline{\eta})$ such that each function of $\underline{\xi}$ appears exactly once among the parametrizations of u with respect to $\underline{\eta}$.

A combinatorial logic network is associated with each Boolean formula. Within a given technological framework, a Boolean formula tends to determine the network associated with it. The same is true of Galois formulas. A logic network will be said to implement, generate, or perform the function represented by the associated formula.

A logic net represents a formula as a graph, primitives of the formula being indicated by nodes of the graph. Such a graph is directed (in the direction of signal flow) and is acyclic (i.e., loop-free). The graph has inputs representing the variables of the formula and an output representing the value of the formula. The number of levels of a graph is the length of the longest path in the graph, exclusive of input nodes, and is independent of the direction of leveling.

Figure 1 illustrates symbols which will consistently represent NOT gates, EXCLUSIVE OR gates, k-input AND gates and k-input OR gates in drawings throughout this report.

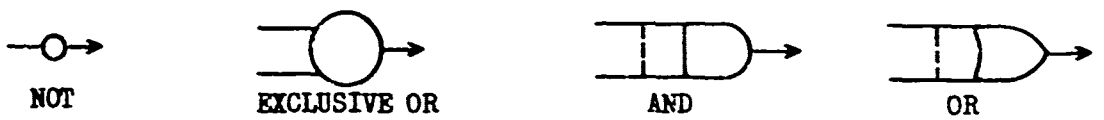


Figure 1. Representation of the NOT Gate, EXCLUSIVE OR Gate, k-Input AND Gate, and k-Input OR Gate

SECTION II

UNIVERSAL FUNCTION THEORY

In this section the concept of a universal function is defined and developed in the full generality of finite relational structures. The interrelationships of universal functions are explored, and they are shown to be universal elements of suitably defined functors.

In the years preceding 1968, government-supported research investigated network optimization and fault-detection and correction for a variety of formulations of Boolean circuitry whose function was determined by external control. This was an effort to anticipate the most important challenge of the future LSI technology to logic design. Such a technology promised vastly reduced marginal cost for circuit complexity with greatly increased initial cost for a circuit design. The ultimate answer to the new cost orientation was the universal module--the network capable of generating an arbitrary function of a fixed set of inputs. Today's read-only memory (ROM) technology owes much to the pioneering support of this research by Air Force Cambridge Research Laboratories.

Through these years the only Boolean function actually investigated in this research activity was that of the selected Shannon decomposition, though the Reed decomposition was known. This work culminated in 1967 with the development of the most efficient means of generating the selected Shannon decomposition in [2].

The first systematic investigation of the theory of universal Boolean functions was published in [3]. There it was established that the Reed and Shannon decompositions were just two of the 2^{2^n} distinct universal functions for n Boolean variables. The following year, based on this work, category theory shed new light on the concept of universal Boolean

functions and hinted at their extension to other mathematical disciplines ([4]). In 1970, under the continuing support of AFCRL, the new approach to logic design utilizing Galois theory was investigated, and a theory of universal Galois functions was developed paralleling those of Boolean algebra. It will now be seen that all of these results arise from the finiteness of a mathematical structure. The theory of universal functions is developed and the interrelationships of universal functions are derived in the full generality of finite mathematical structures. Since, by its nature, every approach to logic design must be based on a complete finite mathematical structure, the theory presented here will be immediately applicable to every possible approach to logic design, past and future.

In this report relational structures of interest are the two-element Boolean algebra and dyadic Galois fields over GF(2). However, the results proven here apply to all serious contenders for the role of supplying the mathematical foundation for logic design. Furthermore, they may be generalized to infinite structures in various ways. Concepts and terminology from category theory are elementary and may be found in the appendix.

Throughout this section, R is assumed to be a fixed relational structure with finite universe U of cardinality c. (A "relational structure" is a mathematical structure with distinguished functions and relations over a universe. Relational structures are a basic concept of model theory and are discussed in texts such as [6] and [7].) A function $u:U^{k+d} \longrightarrow U$ is <u>universal for k variables</u> (with respect to R) if and only if, for every function $f:U^{k} \longrightarrow U$, there exists $\underline{a} \in U^{d}$ such that

$$f(\underline{x}) = u(\underline{x}, \underline{a}).$$

Universal functions have played a quiet but important role in logic design theory. The Shannon or minterm decomposition (disjunctive normal form) for variables $x_0, x_1, \ldots, x_{k-1}$,

$$y_0\bar{x}_0\bar{x}_1\cdots\bar{x}_{k-1} \vee y_1x_0\bar{x}_1\cdots\bar{x}_{k-1} \vee \cdots \vee y_{k*}x_0x_1\cdots x_{k-1}, \qquad (2-1)$$

was the universal function that immediately established the theoretical adequacy of Boolean algebra for combinational logic design and at the same time offered a canonical form for Boolean functions. The maxterm decomposition (conjunctive normal form) is another formulation of the same universal function, while the Reed or ring decomposition,

$$y_0 \oplus y_1 x_0 \oplus y_2 x_1 \oplus y_3 x_0 x_1 \oplus \cdots \oplus y_{k^*} x_0 \cdots x_{k-1}, \qquad (2\text{-}2)$$

results in an entirely distinct universal function for k variables. Reformulation of (2-1) and (2-2) in terms of various logical primitives in accord with shifting technology has supplied the foundation both for the initial approach to the design problem and for various optimization algorithms, such as the Quine-McCluskey method of prime implicants. In addition, universal Boolean functions served as a foundation for the theory of Boolean function decomposition and led the way in various techniques of function composition, such as "treeing" identical modules to obtain more complex or more reliable modules. With a rapidly increasing emphasis on building the maximum of flexibility into a minimum of module types, universal functions begin to assume a greater potential for direct application.

The difference in the form of (2-1) and (2-2) suggests that differing universal functions may be preferred in different technologies [3]. This raises the question of the existence of other universal functions, not only in Boolean algebra but in finite field theory. This section develops the theory of universal functions in the general context of finite relational structures, indicates their interrelationship and shows that they are the universal elements of appropriately defined functors.

Proposition 2.1. If $u: U^{k+d} \longrightarrow U$ is universal for k variables, then d equals c^k.

Proof. There are exactly c^{c^k} k-variable functions defined on U, and there are c^d parametrizations of the final d variables. □

When U is the two-element Boolean algebra B, c is two, and if U is G_n, then c is 2^n. Thus,

Corollary 2.2. If u is universal for k Boolean variables, then u has 2^k selection variables □, and

Corollary 2.3. If u is universal for k variables over $GF(2^n)$, then u possesses 2^{kn} selection variables.□

The following theorem can be viewed as a generalization of the "treeing" technique in logic design.

Theorem 2.4. Let $|\underline{x}| = j$, $|\underline{y}| = k$ and $|\underline{z^{(i)}}| = c^k$, $i = 0, \ldots, c^j-1$. If u is universal for j variables and v is universal for k variables, then $u(\underline{x}, v(\underline{y}, \underline{z^{(0)}}), \ldots, v(\underline{y}, \underline{z^{(c^j-1)}}))$ is universal for $(j + k)$ variables $\underline{x}$, $\underline{y}$.

Proof. Take f a $(j + k)$-argument function. For each k-length sequence $\underline{a}$ of members of U, define $f_{\underline{a}}(\underline{x})$ as the j-argument function $f(\underline{x}, \underline{a})$. By the universality of u, for each k-length sequence $\underline{a}$, there exists a unique c^j-length sequence $\underline{b}$, depending on $\underline{a}$, such that $f_{\underline{a}}(\underline{x}) = u(\underline{x}, \underline{b})$. Define $g: U^k \longrightarrow U^{c^j}$ such that

$$f_{\underline{a}}(\underline{x}) = u(\underline{x}, g(\underline{a})) \tag{2-3}$$

for $\underline{a} \in U^k$. By the universality of v, for each $i \in \underline{c^j}$, there exists a unique c^k-length sequence $\underline{p^{(i)}}$ such that $g_i(\underline{y}) = v(\underline{y}, \underline{p^{(i)}})$. From (2-3) it follows that

$$f(\underline{x}, \underline{y}) = u(\underline{x}, v(\underline{y}, \underline{p^{(0)}}), \ldots, v(\underline{y}, \underline{p^{(c^j-1)}})). \tag{2-4}$$

The uniqueness of the parametrization $(\underline{p^{(0)}}), \ldots, \underline{p^{(c^j-1)}})$ is guaranteed uniqueness specifications for the universality of u and v. This can be seen by supposing that

$$f(\underline{x}, \underline{y}) = u(\underline{x}, v(\underline{y}, \underline{q^{(0)}}), \ldots, v(\underline{y}, \underline{q^{(0^j - 1)}})) \tag{2-5}$$

and replacing $\underline{y}$ by $\underline{b}$ in (2-4) and (2-5) for all possible $\underline{b} \in \underline{c^k}$.□

Figure 2 and Figure 3 illustrate successive applications of Theorem 2.4 to obtain a universal module for ten Boolean variables. In Figure 2 both u and v are universal for two Boolean variables. That is,

$$|\underline{x}| = |\underline{y}| = 2, \text{ and } |\underline{z^{(i)}}| = 4, \; i = 0, 1, 2, 3.$$

Thus, the tree in Figure 2 is universal for four Boolean variables. Figure 3 takes the function generated by the entire tree of Figure 2 and u and takes v as universal for $\underline{y}$. Here,

$$|\underline{x}| = 4 \text{ and } |\underline{y}| = 6, \text{ while}$$

$$|\underline{z}^{(i)}| = 2^6, \quad i = 0, \ldots, 15.$$

Note that here the v-modules of Figure 3 are regarded as universal for six Boolean variables although each has 2^6 function selection lines which are suppressed in the figure.

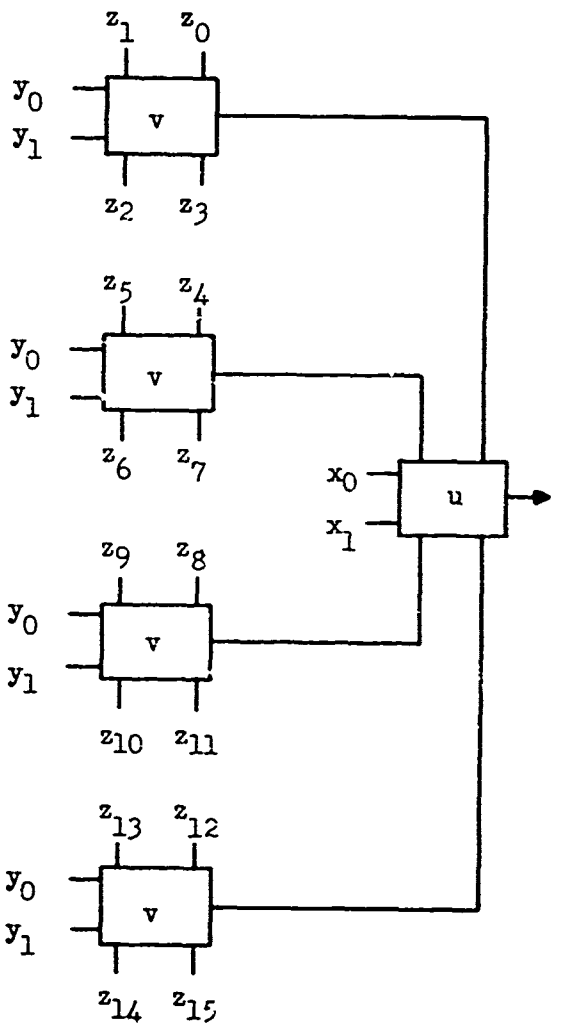


Figure 2. A Universal Tree for Four Variables with Two-Variable Universal Nodes

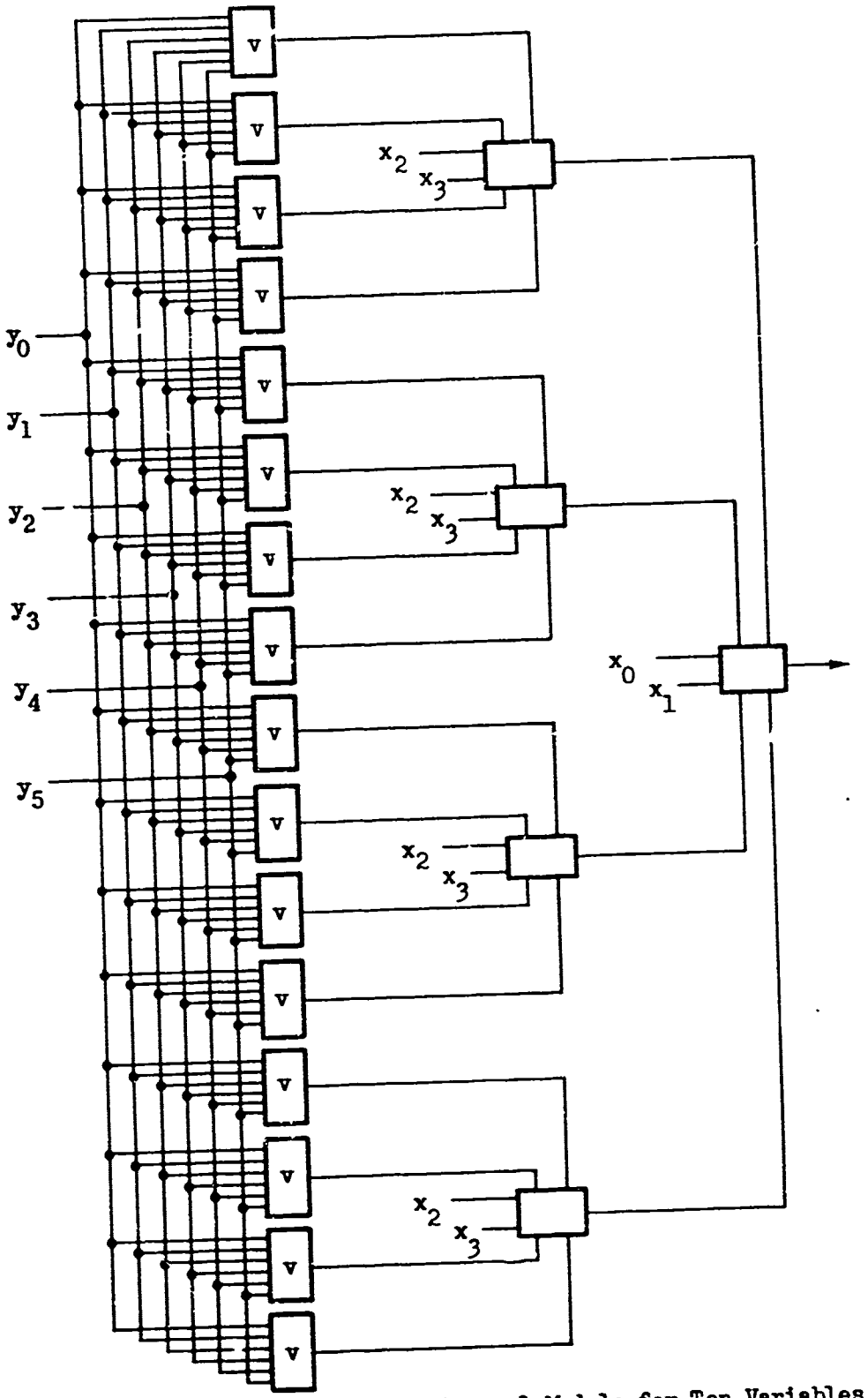


Figure 3. Treeing to Obtain a Universal Module for Ten Variables

The next theorem is a generalization of the basic lemma in Boolean function decomposition theory.

Theorem 2.5. The function u is universal for k variables if and only if for each natural number j and each $(k + j)$-variable function f, there exists a unique function sequence $\underline{g}: U^j \longrightarrow U^{c^k}$ such that

$$f(\underline{x}, \underline{y}) = u(\underline{x}, \underline{g}(\underline{y})).$$

Proof. Suppose u is universal for k variables. The theorem is proven by induction on j. The case j equal zero restates the definition of universality of u, so assume the theorem holds for j equal m, $m \geq 0$, and take f a $(k + m + 1)$-variable function. Let $|\underline{x}| = k$. For each member a of U, define $f_a(\underline{x}, \underline{y}) = f(\underline{x}, \underline{y}, a)$; by the induction hypothesis, there exists a unique function sequence $\underline{g}^a: U^m \longrightarrow U^{c^k}$ such that $f_a(\underline{x}, \underline{y}) = u(\underline{x}, \underline{g}^a(\underline{y}))$. This determines the $(m + 1)$-argument function sequence $\underline{g}$ such that, for each $a \in U$, $\underline{g}(\underline{y}, a) = \underline{g}^a(\underline{y})$, for which $f(\underline{x}, \underline{y}, y_m) = u(\underline{x}, \underline{g}(\underline{y}, y_m))$. The uniqueness of $\underline{g}$ is immediate, completing the induction. The converse is seen by taking j equal zero. □

Reinterpreting the v-modules of Figure 3, Theorem 2.5 implies that for each ten-variable Boolean function f, there exist a unique sequence of six-variable functions $(v_0, \ldots, v_{15})$ such that the network of the figure generates f. In general, if j is zero (i.e., if $\underline{y}$ is the empty sequence), $\underline{g}(\underline{y})$ is a sequence of constants and Theorem 2.5 states that a universal function has a unique parametrization resulting in f.

The following theorem establishes the relationship that exists among universal functions for the same number of variables and, with Theorem 2.4, enables one to obtain all universal functions from a single function universal for one variable.

Theorem 2.6. Let u be universal for k variables. The function v is a universal for k variables if and only if for some function sequence $\underline{g}: U^{c^k} \longleftrightarrow U^{c^k}$,

$$v(\underline{x}, \underline{y}) = u(\underline{x}, \underline{g}(\underline{y})).$$

Proof. Suppose v is universal for k variables. By Theorem 2.5, there exist function sequences $\underline{g}$ and $\underline{h}$, each mapping U^{c^k} into itself, such that $v(\underline{x}, \underline{y}) = u(\underline{x}, \underline{g}(\underline{y}))$ and $u(\underline{x}, \underline{z}) = v(\underline{x}, \underline{h}(\underline{z}))$. From this it follows that $u(\underline{x},\underline{z}) = u(\underline{x}, (\underline{g}\ \underline{h})(\underline{z}))$. By Proposition 2.1 and the uniqueness of $(\underline{g}\ \underline{h})$, this implies that $(\underline{g}\ \underline{h})$ is the identity on U^{c^k}, whence $\underline{g}$ is invertible and therefore a permutation of U^{c^k}.

Suppose now that $\underline{g}$ is a permutation of U^{c^k} and $v(\underline{x}, \underline{y}) = u(\underline{x}, \underline{g}(\underline{y}))$. If f is any k-variable function on U, there exists a unique sequence $\underline{a}$ such that $f(\underline{x}) = u(\underline{x}, \underline{a}) = v(\underline{x}, \underline{g}^{-1}(\underline{a}))$. Any parametrization of v other than $\underline{g}^{-1}(\underline{a})$ resulting in the same function f would violate the uniqueness of the parametrization $\underline{a}$ for u, so v is universal for k variables as required. □

Corollary 2.7. There are exactly $(c^{c^k})!$ universal functions for k variables over a c-element relational structure. □

Corollary 2.8. There are exactly $(2^{2^k})!$ universal functions for k Boolean variables. □

Corollary 2.9. There are exactly $(2^{n2^{kn}})!$ universal functions for k variables over $GF(2^n)$. □

Certain categories are now defined. The first is the category $\mathcal{E}$ whose objects are the Cartesian products U^k, k = 0, 1, ..., with $\hom(U^k, U^j)$ consisting of function sequences $\underline{g}: U^k \longrightarrow U^j$. Next, consider the sequence of categories $\mathcal{K}^m$ (m = 1, 2, ...) whose objects $F_{m,k}$ are the sets of (m + k)-argument functions, k = 0, 1, ... The morphism classes $\hom(F_{m,k}, F_{m,j})$ of $\mathcal{K}^m$ are the sets of transforms T mapping $F_{m,j}$ into $F_{m,k}$.

Let m be a fixed positive integer. There is a contravarient functor $\mathcal{J}^m$ from $\mathcal{E}$ to $\mathcal{K}^m$ defined by:

$\mathcal{J}^m(U^k) = F_{m,k}$, k = 0, 1, ..., and for $\underline{g}: U^k \longrightarrow U^j$, $\mathcal{J}^m(\underline{g}): F_{m,j} \longrightarrow F_{m,k}$ such that, for each member f of $F_{m,k}$,

$$\mathcal{J}^m(\underline{g})(f) = f(\cdot, \underline{g}(\cdot)),\ j = 0, 1, \ldots .$$

Theorem 2.10. The function u is universal for m variables if and only if $u \in \mathfrak{J}^m(U^{c^m})$ is a universal element of $\mathfrak{J}^m$.

Proof. Let u be universal for m variables. By Proposition 2.1, $u \in \mathfrak{J}^m(F_{m,c^m})$. Theorem 2.6 guarantees that for each member f of $\mathfrak{J}^m(U^k)$ there exists a unique function sequence $\underline{h}$ such that

$$f(\underline{x}, \underline{y}) = u(\underline{x}, \underline{h}(\underline{y})) = (\mathfrak{J}^m(\underline{h})u)(\underline{x}, \underline{y}).$$

Thus, $u \in \mathfrak{J}^m(U^{c^m})$ is a universal element of $\mathfrak{J}^m$.

Let u be any universal function for m variables, and let $v \in \mathfrak{J}^m(U^k)$ be a universal element. Then there exists a unique function sequence $\underline{g}: U^{c^m} \longrightarrow U^k$ such that

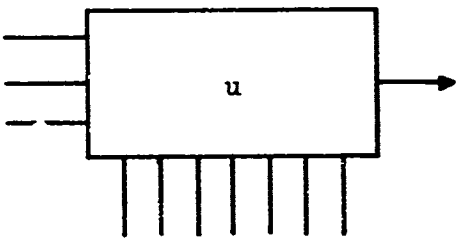
$$u(\underline{x}, \underline{y}) = (\mathfrak{J}^m(\underline{g})v)(\underline{x}, \underline{y}) = v(\underline{x}, \underline{g}(\underline{y})),$$

and there exists a unique function sequence $\underline{h}$ such that

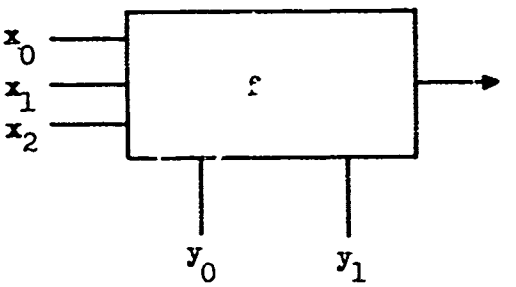
$$v(\underline{x}, \underline{z}) = u(\underline{x}, \underline{h}(\underline{z})) = v(\underline{x}, \underline{g}\,\underline{h}(\underline{z})) = v(\underline{x}, \underline{h}\,\underline{g}(\underline{z})) = (\mathfrak{J}^m(\underline{i}))v)(\underline{x}, \underline{z}),$$

where $\underline{i}$ is the identity function on U^{c^m}. This also implies that $u(\underline{x}, \underline{y}) = u(\underline{x}, \underline{h}\,\underline{g}(\underline{y}))$, from which it follows that $\underline{h}\,\underline{g}$ and $\underline{g}\,\underline{h}$ are both the identity on U^{c^m}. Hence, $\underline{h}$ is a permutation of U^{c^m} and, by Theorem 2.6, v is a universal function. □

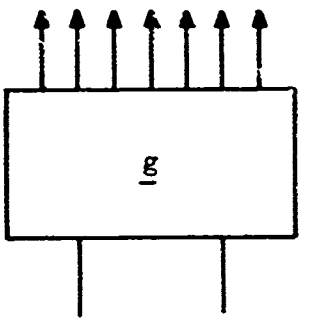
Figure 4 illustrates the activity of a universal Boolean function as a universal element.



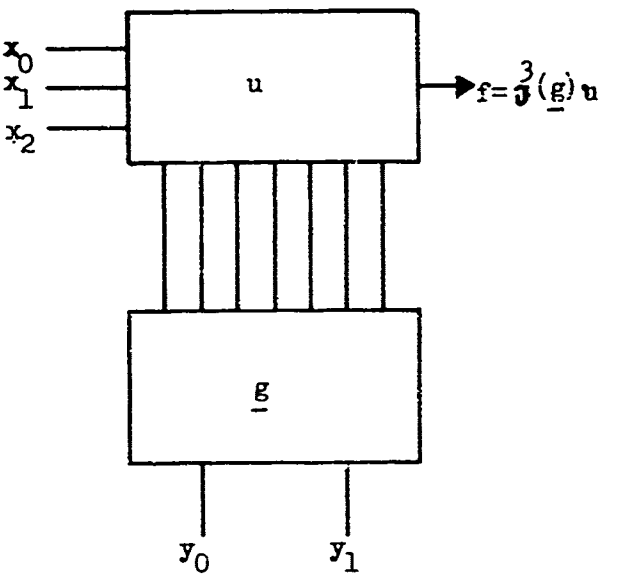
a. The Universal Function u Is Given.



b. The Desired Function $f(x_0, x_1, x_2, y_0, y_1)$ Is Determined.

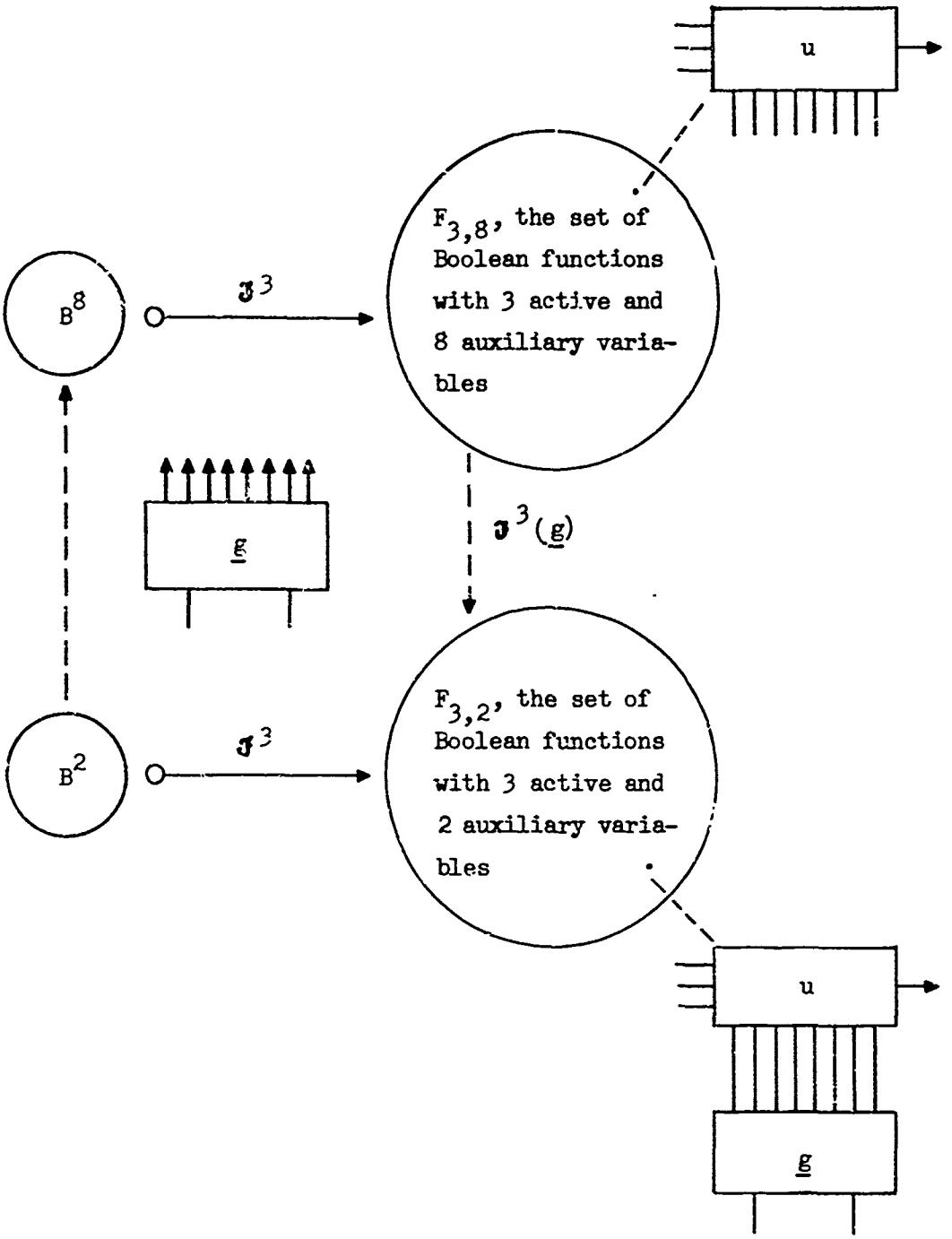


c. The Suitable Translator Is Found.



d. $f(x_0, x_1, x_2, y_0, y_1)$ as Obtained from u.

Figure 4. Use of a Universal Boolean Function as a Universal Element (Sheet 1 of 2)



e. Diagrammatic Summary of the Above Process.

Figure 4. Use of a Universal Boolean Function as a Universal Element (Sheet 2 of 2)

SECTION III

GALOIS LATTICE-LIKE OPERATIONS

The dyadic Galois fields $GF(2^n)$ with Galois addition and Galois multiplication can be regarded as generalizations of the two-element Boolean ring, the field with primitives EXCLUSIVE OR and AND. Galois addition and Galois multiplication are the Galois analogues of Boolean EXCLUSIVE OR and AND, respectively. One question which presents itself is: are there natural Galois analogues of other Boolean primitives? In particular, what are the Galois analogues of the Boolean lattice operations: AND, OR, NOT? Answers to this question could offer competing frameworks for logic design in an MSI/LSI environment and Galois structures whose algebra is closer to that of the traditional Boolean approach. This section offers sets of lattice-like Galois operations, including one set which gives a true lattice.

The following is a list of basic identities for the lattice operations of Boolean algebra, with '$\vee$' the disjunction operator, '$\wedge$' the conjunction operator and overlining the inversion operator. As a set of axioms, the list is complete but redundant.

(a)	(b)	
$\overline{0} = 1$	$\overline{1} = 0$	(3-1)
$x \wedge 0 = 0$	$x \vee 1 = 1$	(3-2)
$x \wedge 1 = x$	$x \vee 0 = x$	(3-3)
$x \wedge \overline{x} = 0$	$x \vee \overline{x} = 1$	(3-4)
$\overline{\overline{x}} = x$		(3-5)
$x \wedge x = x$	$x \vee x = x$	(3-6)
$\overline{x \wedge y} = \overline{x} \vee \overline{y}$	$\overline{x \vee y} = \overline{x} \wedge \overline{y}$	(3-7)

(a)	(b)	
$x \wedge y = y \wedge x$	$x \vee y = y \vee x$	(3-8)
$x \wedge (y \wedge z) = (x \wedge y) \wedge z$	$x \vee (y \vee z) = (x \vee y) \vee z$	(3-9)
$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$	$x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$	(3-10)

The first approach to defining a Galois AND, Galois OR and Galois NOT is as follows:

$$\xi \wedge \eta = \xi\eta, \tag{3-11}$$

$$\xi \vee \eta = \xi + \eta + \xi\eta, \tag{3-12}$$

$$\overline{\xi} = \xi. \tag{3-13}$$

In this case, all the identities above hold for $GF(2^n)$ except for (3-4), (3-6) and (3-10). Thus, while under these operations the Galois field has many of the features of a lattice, it is not a lattice. The failure here seems to be rooted in the fact that Galois NOT is uninterpretable as a complementary set (3-4) and that Galois AND and OR are not true reducing and collecting operators, as indicated by the failure of the idempotency laws (3-6) and DeMorgan's laws (3-10). Enough of the lattice structure survives to make meaningful use of some lattice intuition in utilizing (3-11), (3-12), and (3-13). Implementation of (3-13) for additive codes is just bitwise inversion, as illustrated in Figure 5. Figure 6 displays the Galois OR gate for (3-12).

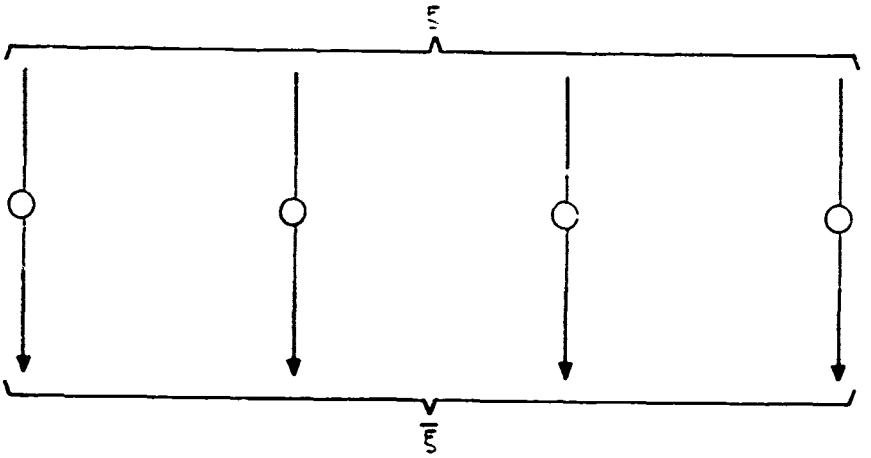


Figure 5. A Galois NOT Gate for $GF(2^4)$

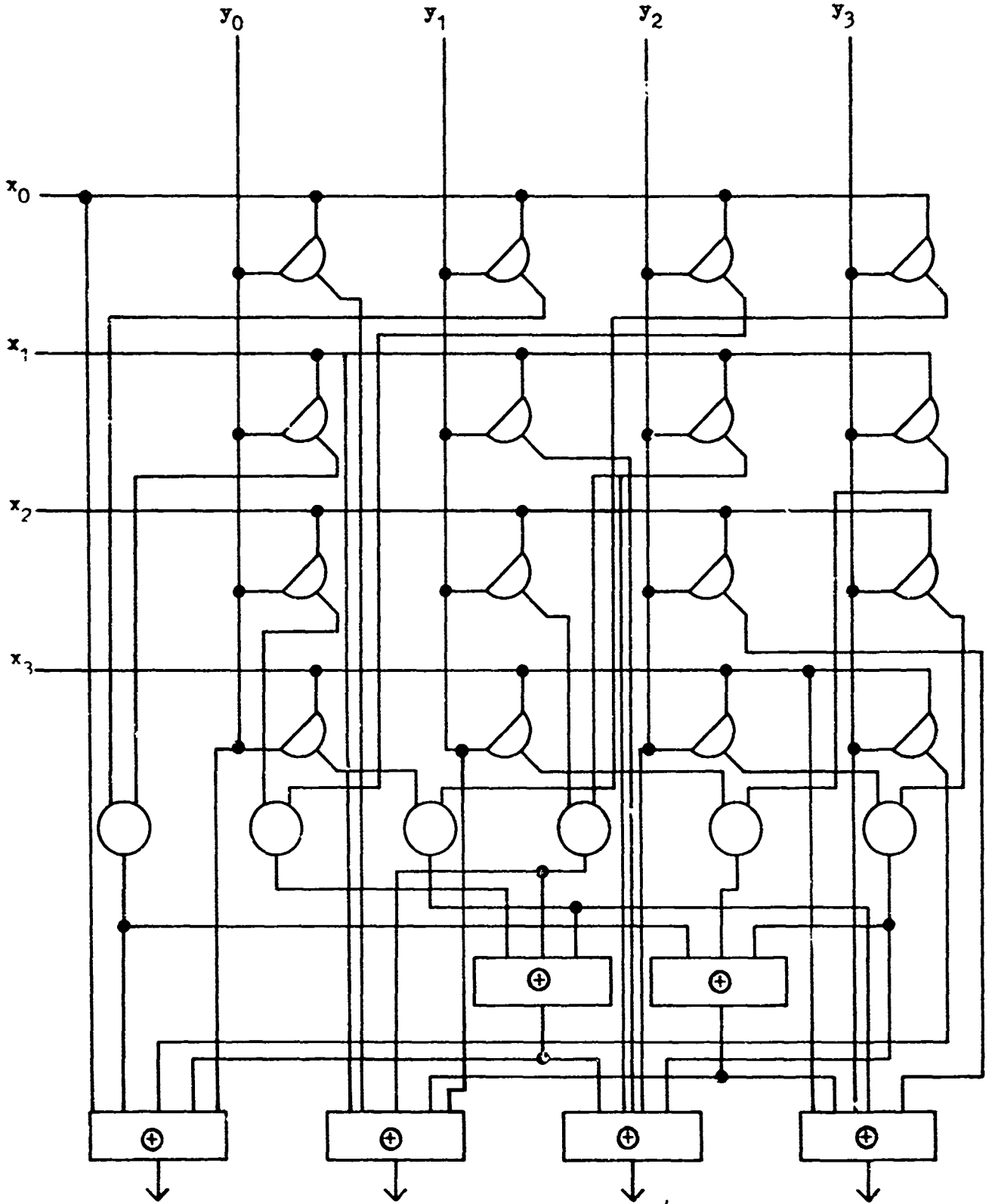


Figure 6. A Galois OR Gate for GF(2^4)

Note that (3-12) mixes first and second order terms. One might hold this feature responsible for some of the failures above and try to elimnate them by defining

$$\xi \overset{\prime}{\vee} \eta = \xi + \eta + \sqrt{\xi\eta}. \tag{3-14}$$

This definition salvages (3-6b) and (3-10a) but only at the expense of (3-24), (3-7) and (3-9a).

Part of the purpose in seeing how closely one can approximate lattice operations in Galois structures lies in the desirability of building up Galois functions from specific conditions much as Boolean functions can be generated by collecting minterms on which the function is nonzero. Characteristic functions provide a foundation for such development. The characteristic function for the sequence $\underline{\alpha}$ of Galois values is the function $\chi_{\underline{\alpha}}(\underline{\xi})$, defined

$$\chi_{\underline{\alpha}}(\underline{\xi}) = \begin{cases} 0 \text{ if } \underline{\xi} = \underline{\alpha}, \\ \\ 1 \text{ if } \underline{\xi} \neq \underline{\alpha} \end{cases} \tag{3-15}$$

Like all other Galois functions, characteristic functions can be written as polynomials. Here,

$$\chi_{\underline{\alpha}}(\underline{\xi}) = \prod_{0}^{|\underline{\xi}|-1} [1 + (\xi_i + \alpha_i)^{n^*}]$$

From (3-15), it is clear that every Galois function of one Galois variable ξ can be written

$$f(\xi) = \sum_{\alpha \in G_n} f(\alpha)\chi_{\alpha}(\xi)$$

and that every Galois function $f(\underline{\xi})$ of the sequence of k variables $\underline{\xi}$ can be written

$$f(\underline{\xi}) = \sum_{\underline{\alpha} \in G_n^k} f(\underline{\alpha}) \chi_{\underline{\alpha}}(\underline{\xi}).$$

A true lattice structure can be defined on $GF(2^n)$ which makes characteristic functions the atoms of a distributive Galois lattice. Let $\underline{C}$ be the code for $GF(2^n)$, and define the set transform $S: GF(2^n) \longrightarrow \underline{n}$ such that, for each $\alpha \in GF(2^n)$,

$$S(\alpha) = \{i : C_i \alpha = 1\}$$

Now define

$$\bar{\xi} = S^{-1}(\underline{n} - S(\xi)), \tag{3-16}$$

$$\xi \vee \eta = S^{-1}(S\xi \cup S\eta), \text{ and} \tag{3-17}$$

$$\xi \wedge \eta = S^{-1}(S\xi \cap S\eta). \tag{3-18}$$

The structure $\langle GF(2^n); \vee, \wedge, \bar{\cdot} \rangle$ is a distributive lattice, with features corresponding closely to the two-value Boolean lattice structure. In particular, the formula

$$f(\underline{\xi}) = \bigvee_{\underline{\alpha} \in G_n^k} f(\underline{\alpha}) \chi_{\underline{\alpha}}(\underline{\xi}).$$

for k Galois variables $\underline{\xi}$ is the Galois analogue of

$$f(\underline{x}) = \bigvee_{0}^{n^*} y_i \mu_i(\underline{x})$$

the selected minterm formula for k Boolean variables $\underline{x}$.

As a lattice, the Galois operations (3-16), (3-17) and (3-18) are the most convenient from a mathematical point of view. Unfortunately, they would not seem to be good choices for an MSI/LSI technology because their implementations grow linearly with n instead of n^2, resulting in poor gate-to-pin ratios.

SECTION IV

SPECIAL NETWORKS

This section discusses two techniques utilizing networks related to Galois logic design. Each offers a specific network, depending both on the power of the Galois field and the selected code. The first is a method of enhancing a Galois multiplication gate to perform binary addition, while the second offers an improved version of the linear Galois gate.

Let γ be a generator of $GF(2^n)$, and let $\underline{L}$ be the logarithmic code

$$\underline{L}(0) = (1, 1, \ldots, 1),$$

$$\underline{L}(\gamma^i) = \underline{B}(i),\ i \in \underline{n}^*,$$

where $\underline{B}(i)$ is the reverse binary expansion of i. Galois multiplication has the property that

$$0 \cdot x = x \cdot 0 = 0, \text{ for all } x, \text{ and}$$

$$\gamma^i \gamma^j = \gamma^{(i+j) \bmod n^*},\ i, j \in \underline{n}^*.$$

Thus, a Galois multiplication gate for code $\underline{L}$ performs the function

$$\underline{A}(\underline{1}, \underline{B}(1)) = \underline{A}(\underline{B}(1), \underline{1}), \text{ and}$$

$$\underline{A}(\underline{B}(i), \underline{B}(j)) = \underline{B}(i + j \bmod n^*), i, j \in \underline{n}^*,$$

where "1" denotes the ones sequence and $\underline{A}$ is an n-bit adder with end-around carry for integers written in (reverse) binary notation.

Now let $\underline{C}$ be any code for $GF(2^n)$, and define translators

$$\underline{S} = \underline{C}\,\underline{L}^{-1} \text{ and } \underline{T} = \underline{L}\,\underline{C}^{-1}. \tag{4-1}$$

If $\underline{M}$ is the Boolean function sequence performed by a multiplication gate for $\underline{C}$, then

$$\underline{B}(i + j \bmod n^*) = \underline{L}(\gamma^{i+j \bmod n^*}) = \underline{T}\,\underline{C}(\gamma^{i+j \bmod n^*})$$

$$= \underline{T}\,\underline{M}(\underline{C}(\gamma^i), \underline{C}(\gamma^j)) = \underline{T}\,\underline{M}(\underline{S}\,\underline{L}(\gamma^i), \underline{S}\,\underline{L}(\gamma^j))$$

$$= \underline{T}\,\underline{M}(\underline{S}\,\underline{B}(i), \underline{S}\,\underline{B}(j)),\ i, j \in \underline{n^*}.$$

Figure 7 illustrates the general network suggested by the equation for a four-bit adder with end-around carry.

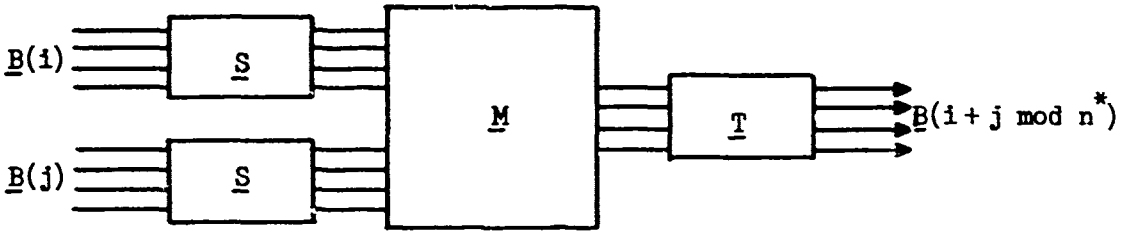


Figure 7. Use of a Galois Multiplication Gate for an Adder with End-Around Carry

An example of this method of implementing four-bit adder with end-around carry is now developed with code $\underline{C}$ given in Table I. In this case (4-1) is satisfied by the logic networks of Figures 8 and 9, while a Galois multiplication gate performing $\underline{M}$ for code $\underline{C}$ is illustrated in Figure 10. Other choices of codes may result in simplifications of networks for $\underline{S}$, $\underline{T}$ and/or $\underline{M}$.

Table I. A Code $\underline{C}$ For GF(2^4)

x	0	1	γ	γ^2	γ^3	γ^4	γ^5	γ^6
$\underline{x}$	0000	1111	1000	0100	1101	0010	1010	1110
x	γ^7	γ^8	γ^9	γ^{10}	γ^{11}	γ^{12}	γ^{13}	γ^{14}
$\underline{x}$	0011	0001	1011	0101	0110	0111	1100	1001

The second technique to be mentioned here is an improved version of the Galois linear gate discussed and utilized in [5]. An implementation of the Galois linear function $(y + zx)$ was offered there consisting of a Galois multiplication gate and Galois addition gate. It was natural to expect that direct implementation of the linear function would result in some internal saving. It is now seen that the internal hardware is reduced slightly, to a bit more than that of a Galois multiplication gate in the forthright manner indicated in Figures 10 and 11 for code $\underline{G}$ given in Table II.

Table II. Geometric Code $\underline{G}$ for GF(2^4)

x	0	1	γ	γ^2	γ^3	γ^4	γ^5	γ^6
$\underline{G}x$	0000	1111	1110	1101	1000	0111	1001	0100
x	γ^7	γ^8	γ^9	γ^{10}	γ^{11}	γ^{12}	γ^{13}	γ^{14}
$\underline{G}x$	1100	1011	0010	0110	1010	0001	0011	0101

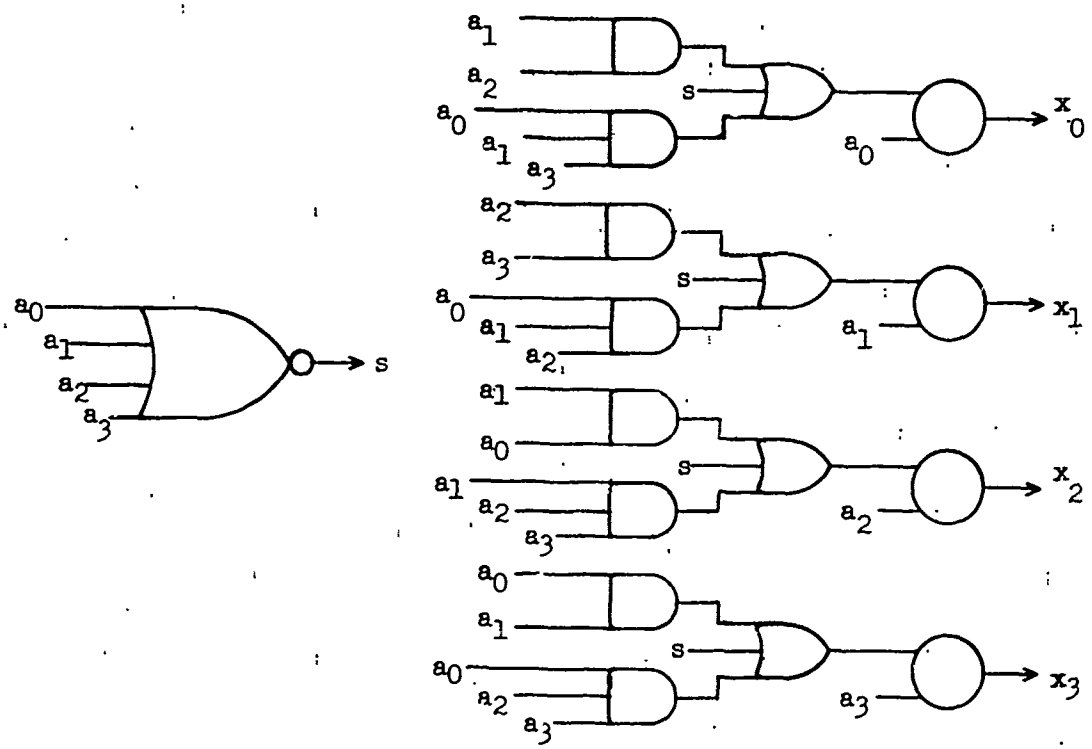


Figure 8. Translator S for Code C

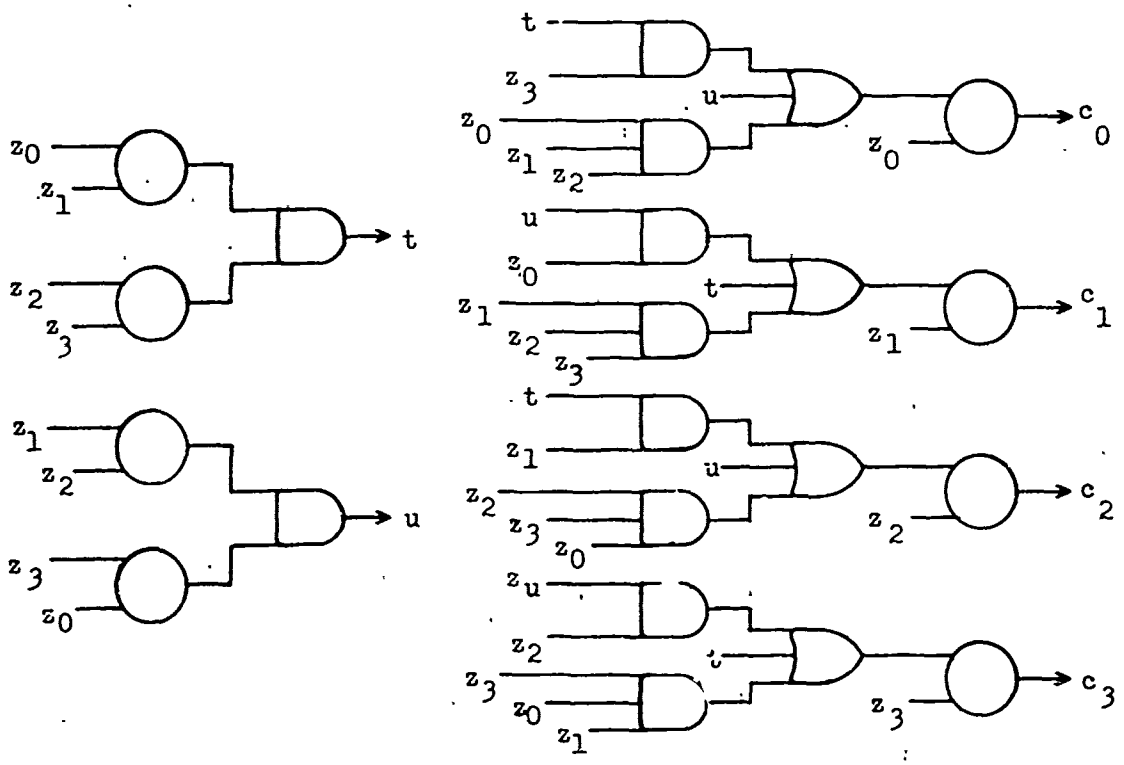


Figure 9. Translator T for Code C

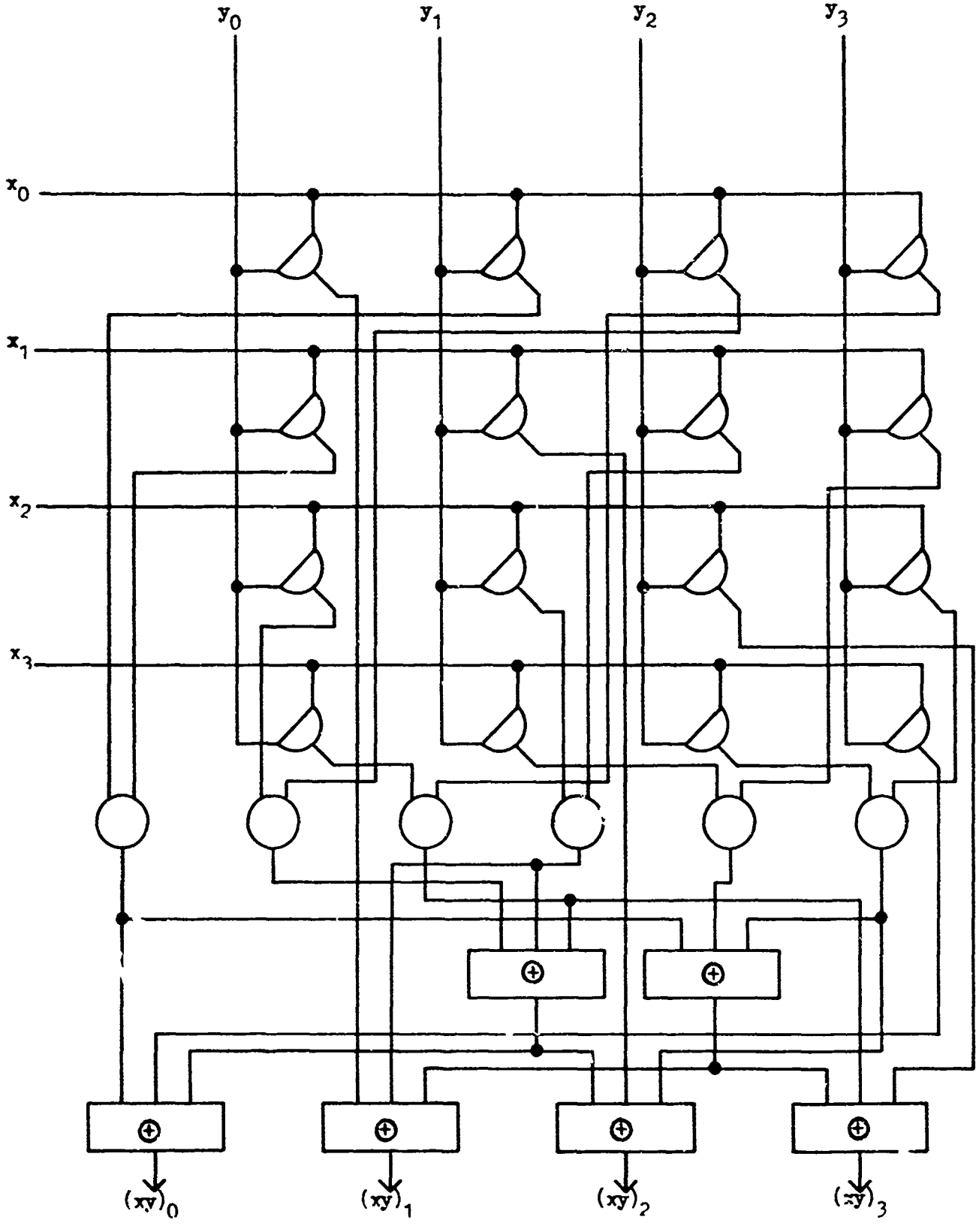


Figure 10. Galois Multiplication Gate for Code C

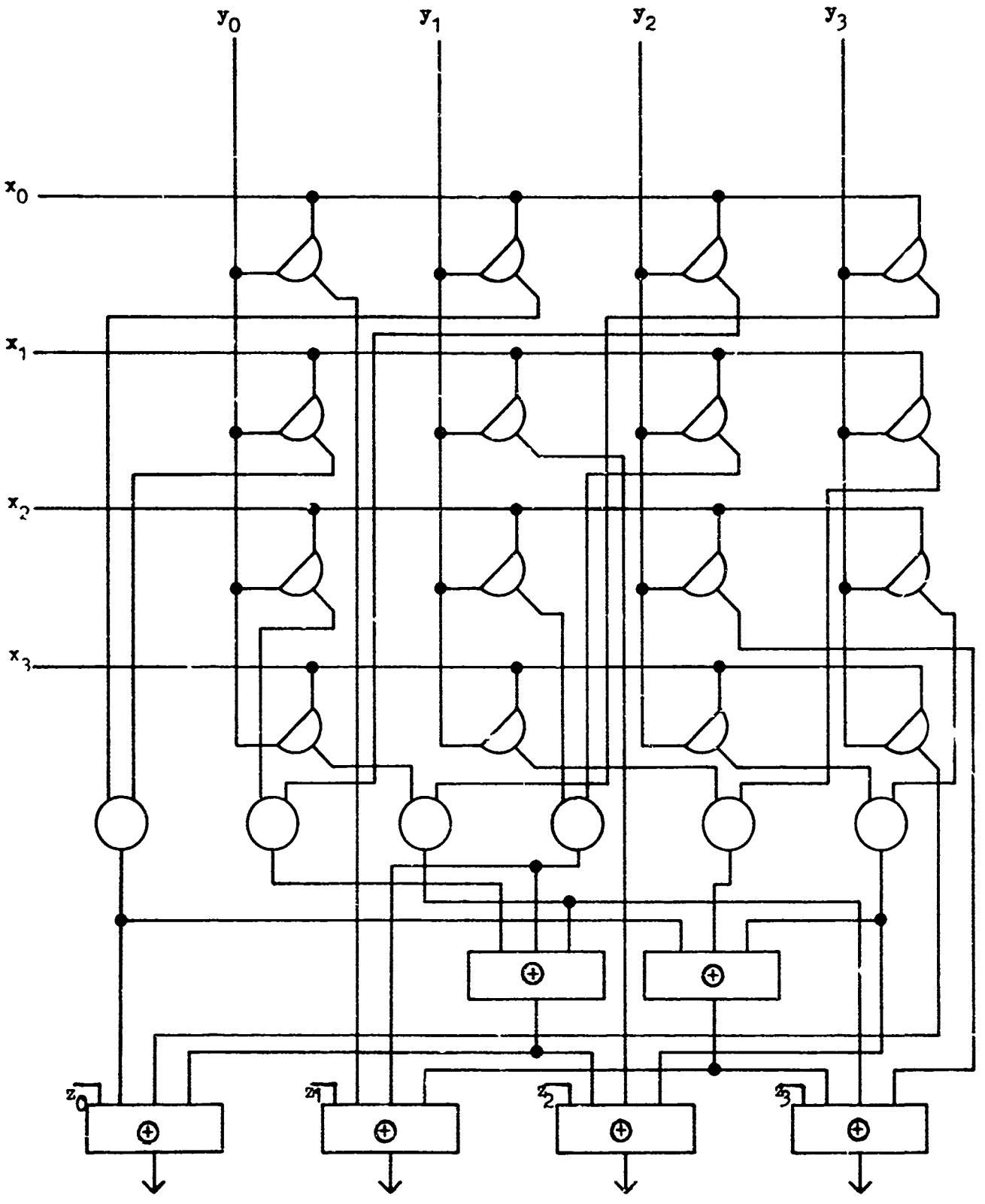


Figure 11. An Improved Linear Module for GF(2^4)

It is evident that the hardware saving from this technique grows at most linearly with the power n of the field while the complexity of the Galois multiplication gate increases with n^2. Thus, one cannot expect significant proportional savings in hardware for large values of n, though switching speed may be significantly enhanced for any value of n by saving a level of hardware. Nevertheless, in view of the minimal difference in complexity between the Galois multiplication gate and its enhancement to a linear gate, it seems unlikely any further improvements will be found. Thus, for additive codes, linear gates of this type may be regarded as optimal, given the optimality of the Galois multiplication gate utilized. Another corollary to this technique is the fact that, due to their minimal difference in gate count, the choice between the plus-times system of [5] and the linear system must be made on the basis of the plus-times requirement of two primitive gates, each with 3n pins, compared to the single 4n-pin primitive gate required for the linear system.

SECTION V

GENERALIZED LINEAR NETWORKS

This section considers a generalization of the linear module and its use in hardware reduction. A linear function of dimension d is a Galois function of the form

$$\eta_0\xi_0 + \eta_1\xi_1 + \cdots + \eta_{d-2}\xi_{d-2} + \eta_{d-1}. \qquad (5\text{-}1)$$

A general linear function is a linear function of unspecified dimension. Used without dimensional specification, the term 'linear function' is reserved for the linear function of dimension two. A nonfactor cascade of linear modules is a cascade of linear modules in which the output of each linear module is the nonfactor input of its successor.

Proposition 5.1. A linear function of dimension d is generated by a nonfactor cascade of d linear modules. □

Figure 12 illustrates the method of generating the d-dimensional linear function by means of a cascade of d linear modules.

A choice set of inputs to a d-dimensional linear function (5-1) is any subset S of d inputs such that for each i (i = 0, ..., d - 2) either ξ_i or η_i is a member of S. Let N be a network of general linear modules without internal fanout. The external inputs of N are those module-inputs which do not arise from previous levels. The network N is called immediate if every general linear module in N has a choice set of external inputs.

In [5] the following theorem was proven establishing the function of a network of linear modules to be largely independent of its geometry.

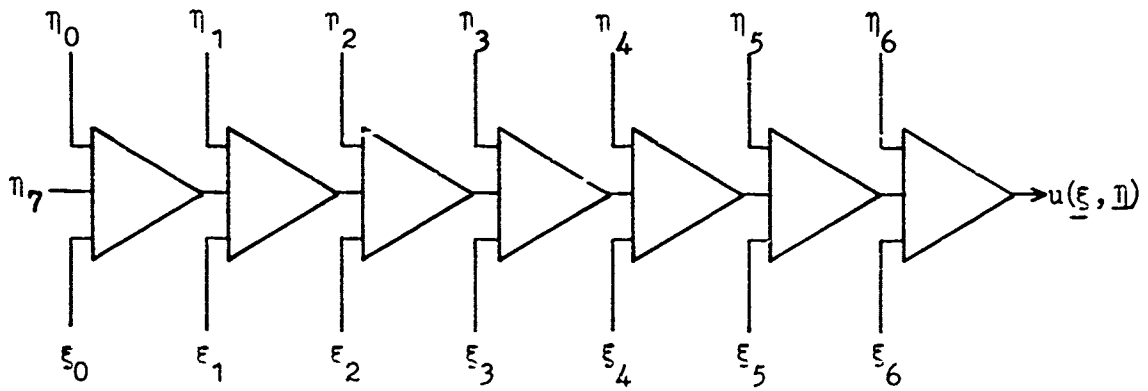


Figure 12. The Nonfactor Linear Cascade of Dimension Seven

Theorem 5.2. If N is a (k - 1)-node immediate network in the linear system, there exists an appropriate sequence of selection variables and powers of ξ to be introduced at the external inputs of N which enables N to perform a general polynomial of length k. □

Now a similar theorem is proven for networks of general linear modules. Define the dimension of a network of general linear modules as the sum of the dimensions of its modules.

Theorem 5.3. Let N be a (k-1)-dimensional immediate network of general linear modules. There exists an appropriate sequence of selection variables and powers of ξ to be introduced at the external inputs of N which enables N to perform a general polynomial of length k.

Proof. By Proposition 5.1, each general linear module of dimension d in N may be replaced by a cascade of d linear modules. If N' is the network resulting from N by replacing each general linear module by such a cascade, then the total number of linear modules in N' is just (k - 1), and N' is a

$(k-1)$-node immediate network of linear modules. By Theorem 5.2, there is an appropriate sequence of selection variables and powers of ξ which can be introduced at the external inputs of N' which enables N' to perform a general polynomial of length k. The same sequence of inputs applied to corresponding points of N then establishes the theorem. □

The effect of the above theorem is to suggest minimally redundant, fault-tolerant arrays similar to those of [5], utilizing d-dimensional linear modules. This permits the use of arbitrarily complex cells in the array without a corresponding increase in field power. If on the other hand one wishes to reduce hardware, Figure 13 illustrates a means of reducing the d-dimensional linear gate to a single linear module. In general, hardware required for a d-dimensional linear gate can be reduced by a factor of approximately $\lfloor d/k \rfloor$ at the cost of a $\lceil d/k \rceil$-increase in switching time.

A similar network, consisting of a single linear module and a unit delay, can be used to generate a general polynomial of length k in $(k-2)$ clock pulses, as in Figure 14.

A general polynomial module of length k, together with a linear module and a t delay, generates a general polynomial of length tk in $t-1$ clock pulses. For

$$\underline{\eta}^{(i)} = (\eta_{2^n-ik}, \cdots, \eta_{2^n-(i-1)k-1}),$$

$$f_i = \eta_{(t-i)k} + \cdots + \eta_{tk-1}\xi^{ik-1},$$

$$f_t = \eta_0 + \cdots + \eta_{tk-1}\xi^{tk-1}.$$

Thus, for $t = \lceil 2^n/k \rceil$, the sequential network of Figure 15 is universal.

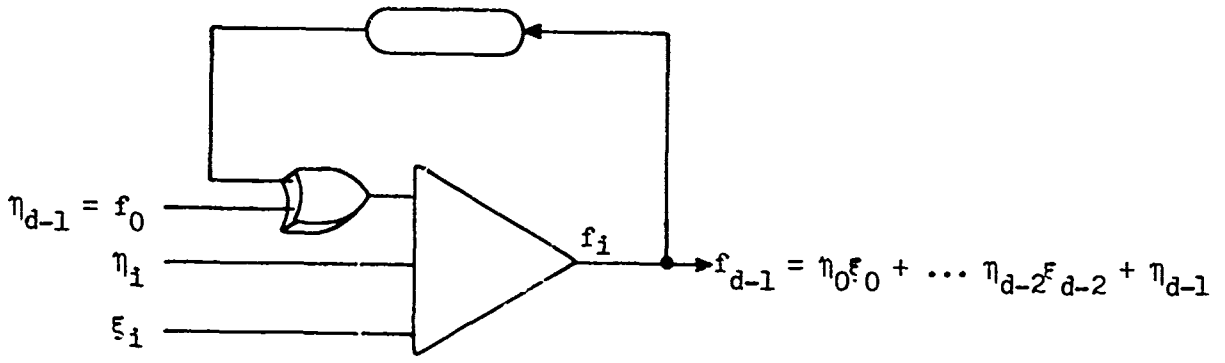


Figure 13. Sequential Implementation of a General Linear Gate

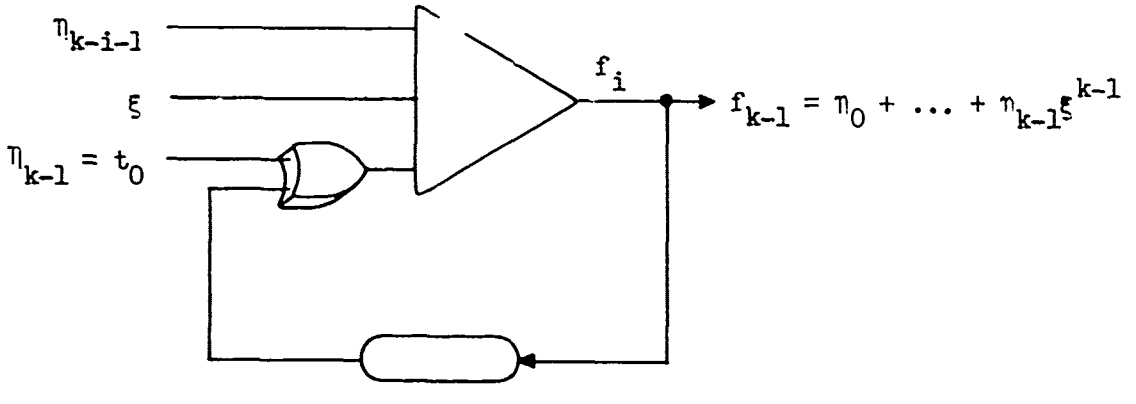


Figure 14. Sequential Implementation of a General Polynomial from a Linear Module

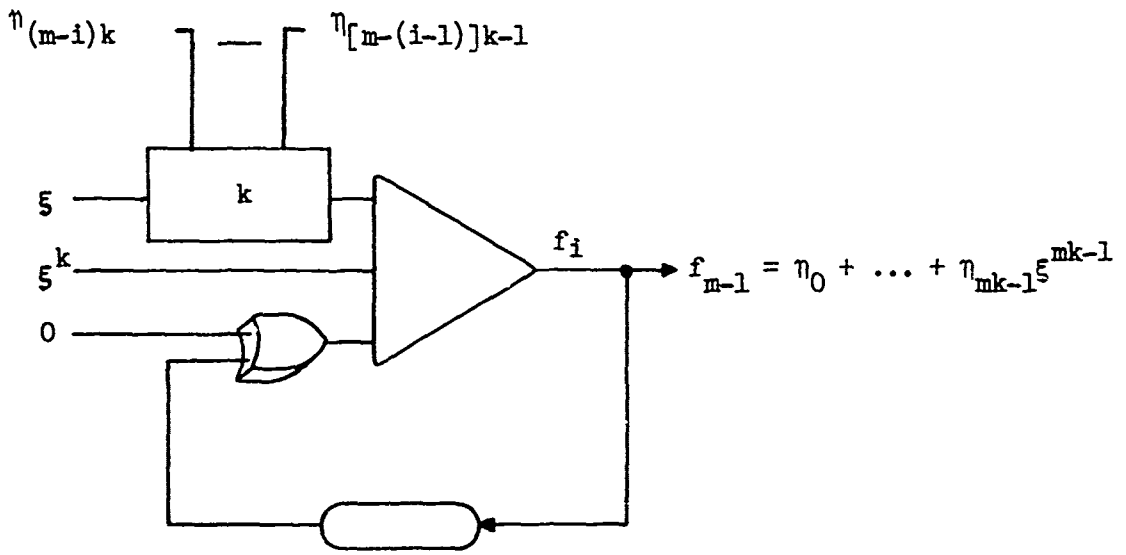


Figure 15. Sequential Implementation of a General Polynomial from a Linear Module and a General Polynomial of Fractional Length

SECTION VI

FUTURE PLANS

Research in Galois logic design has, till the present, been primarily concerned with various primitive gates and their synthesis. It is possible now to interrelate choice of code and primitive complexity. Varieties of primitive sets have been considered, such as the plus-times system, the plus-times hybrid [5], the linear system, Galois pseudolattice systems (Section Three), general linear primitives (Section Five), etc. Furthermore, methods of arriving at Galois functions satisfying Boolean truth tables are known ([5]). To make Galois design practical, two major areas remain to be developed.

First, it is necessary to examine the effect of code choice on the complexity of the Galois polynomial. Similarly, when the number of Boolean functions to be developed on a set of Boolean arguments is less than the power of the overall Galois field, how does the determination of don't-care conditions affect the Galois polynomial? Heuristic answers to these questions may be obtained by means of computer assistance. It is hoped that they will point in the direction of simpler less expensive Galois networks by means of systematic, judicious selection of code and don't-care resolutions.

Second, once the Galois polynomial is available, how is it simplified to reduce gate-count? In principal, a circuit may require a very large number of Galois gates, even for relatively small field power. It is essential to reduce this number by mears of factorization and nesting techniques. So far, research in Galois polynomials has concentrated on deriving polynomial roots, due to the influence of coding theory. This objective seems far less relevant to logic design in that it ignores

function decomposition. It is rather clear that the power of function decomposition algorithms must be brought to bear on the artifically inflated gate-counts of the initial Galois polynomial. It is anticipated that this will be a major effort, but such algorithms are essential to the establishment of the practicality of Galois logic design.

APPENDIX

Like set theory, category theory provides the vocabulary and basic structure as a universe of discourse for whatever discipline has been placed in its setting. Unlike set theory, instead of studying the internal structure of individual objects, category theory focuses attention upon the relations and functions existing between them.

Let $\mathbf{X}$ be a class C of objects together with two functions, hom and $\cdot$, satisfying the following conditions.

C1. For each pair of objects X, Y in C, hom (X, Y) is a family of elements, called morphisms.

C2. For each triple of objects X, Y, Z in C, if f is a member of hom (X, Y) and g is a member of hom (Y, Z) then $g \cdot f$ is a member of hom (X, Z).

If f is a member of hom (X, Y), then f is said to have domain X and codomain Y, all of which is indicated by writing "$f:X \longrightarrow Y$". Whenever it exists, $g \cdot f$ is called the composite of g with f.

$\mathbf{X} = (C, \text{hom}, \cdot)$ will be called a category if, in addition, the following two axioms hold.

C3. The operation $\cdot$ is associative; that is, for W, X, Y, Z in C, if $f:W \longrightarrow X$, $g:X \longrightarrow Y$ and $h:Y \longrightarrow Z$, then $h \cdot (g \cdot f) = (h \cdot g) \cdot f$.

C4. For each object X in C, there is a two-sided identity 1_X with respect to the composition operation; that is, there exists a morphism $1_X:X \longrightarrow X$ such that, for each Y and each $f:Y \longrightarrow X$ and $g:X \longrightarrow Y$, $1_X \cdot f = f$ and $g \cdot 1_X = g$.

A <u>functor</u> $\mathfrak{J}$ from category $\mathcal{K} = (C, \text{hom}, \cdot)$ to category $\mathcal{K}' = (C', \text{hom}', \cdot)$ has the following properties:

F1. $\mathfrak{J}$ is a function from C to C';

F2a. for each pair of objects X, Y in C, if $f: X \longrightarrow Y$, then $\mathfrak{J}(f): \mathfrak{J}(X) \longrightarrow \mathfrak{J}(Y)$, or

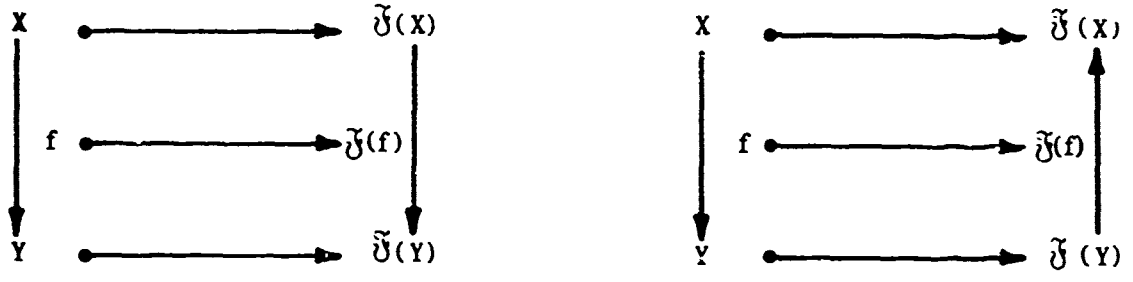
F2b. for each pair of objects X, Y in C, if $f: X \longrightarrow Y$, then $\mathfrak{J}(f): \mathfrak{J}(Y) \longrightarrow \mathfrak{J}(X)$;

F3a. for each pair of morphisms f, g of $\mathcal{K}$ for which g·f is defined, $\mathfrak{J}(g \cdot f) = \mathfrak{J}(g) \cdot \mathfrak{J}(f)$, or

F3b. for each pair of morphisms f, g of $\mathcal{K}$ for which g·f is defined, $\mathfrak{J}(g \cdot f) = \mathfrak{J}(f) \cdot \mathfrak{J}(g)$;

F4. for each object X in C, $1_X = 1_{\mathfrak{J}(X)}$.

The functor is <u>covariant</u> if it satisfies (F2a) and (F3a), <u>contravariant</u> if it satisfies (F2b) and (F3b). (See Figures 16 and 17.)



a. Covariant Functors

b. Contravariant Functors

Figure 16. Typical Behavior of Functors

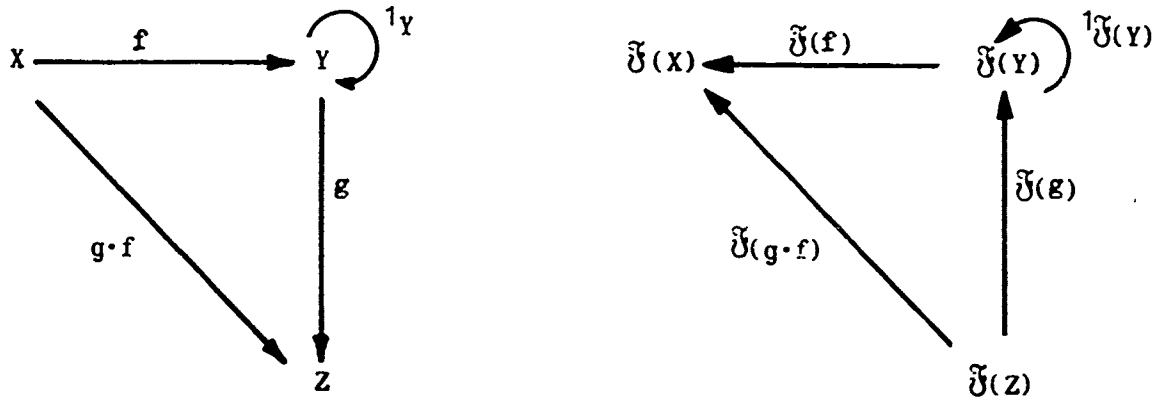


Figure 17. Homomorphic Behavior of a Contravariant Functor

The last concept to be introduced is that of a universal element. A <u>universal element of a contravariant functor</u> $\mathfrak{J}$ on a category $\mathfrak{K}$ is an object R and an element u of $\mathfrak{J}(R)$ with the property that, for each object X of $\mathfrak{K}$ and each member v of $\mathfrak{J}(X)$, there is exactly one morphism $h: X \rightarrow R$ such that

$$\mathfrak{J}(h)u = v.$$

(See Figure 18.)

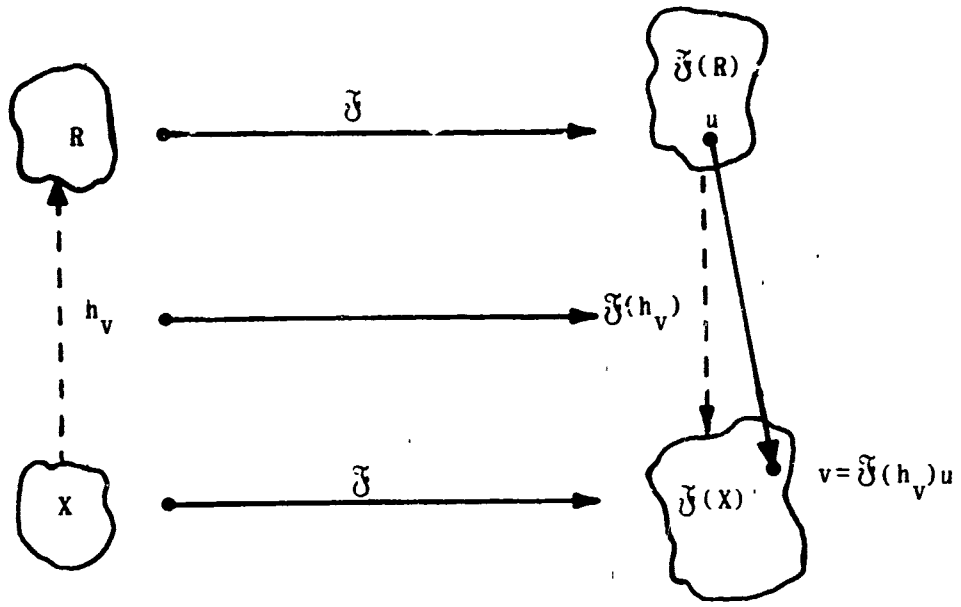


Figure 18. Universal Element $u \in \mathfrak{F}(R)$ for a Contravariant Functor

REFERENCES

[1] Dunning, M., Kolman, B., Investigation of Logic Circuit Complexes, Reliability and Fault Masking in Homogeneous Logical Systems, Univac Division of Sperry Rand Corporation, Final Report to Air Force Cambridge Research Laboratories under Contract AF19(628)-4051, April 1966.

[2] Ellison, J., Kolman, B., and Schiavo, A., Investigation of Logic Circuit Complexes, Universal Function Modules, Univac Division of Sperry Rand Corporation, Final Report to Air Force Cambridge Research Laboratories, Contract No. AF19(628)-6012, 15 April 1967.

[3] Ellison, J. T., Universal Functions, Univac Division of Sperry Rand Corporation, Scientific Report No. 1 to Air Force Cambridge Research Laboratories, Contract No. F19628-67-C-0386, April 1968.

[4] Ellison, J. T., Fan-In Constrained Logic Networks, Univac Division of Sperry Rand Corporation, Final Report to Air Force Cambridge Research Laboratories, Contract No. F19628-67-C-0386, February 1969.

[5] Ellison, J. T. and B. Kolman, Galois Logic Design, Univac Division of Sperry Rand Corporation, Final Report to Air Force Cambridge Research Laboratories, Contract No. F19628-70-C-0067, October 1970.

[6] Robinson, Abraham, Introduction to Model Theory and to the Metamathematics of Algebra, North-Holland Publishing Company, Amsterdam, 1963.

[7] Addison, J. W., Henkin, Leon, and Tarski, Alfred, The Theory of Models, North-Holland Publishing Company, Amsterdam, 1965.

[8] Halmos, Paul R., Lectures on Boolean Algebras, D. Van Nostrand Co., Princeton, New Jersey, 1963.

[9] MacLane, S. and Birkhoff, G., Algebra, The MacMillan Company, New York, 1967.

[10] Laws, Jr., B. A., and Rushforth, C. K., "A Cellular-Array Multiplier for GF(2^m)", IEEE Transactions on Computers, December 1971.